Tool Demo: Browsing Software Repositories

Steven P. Reiss

Department of Computer Science
Brown University
Providence, RI. 02912 USA
spr@cs.brown.edu

Abstract—**We demonstrate a tool for browsing large software repositories such as Github or SourceForge using all the facilities one normally associates with an integrated development environment. The tool integrates code search engines with the Code Bubbles development environment. It lets the user perform and compare multiple searches, investigate and explore the results that are returned, expand searches as necessary, and eventually export appropriate results.**

Keywords—Code search; software repositories; integrated development environments.

INTRODUCTION

An enormous amount of open source code is available today. One of the advantages of having such repositories is that, for almost any small coding problem you need to solve, someone has probably solved the problem or one like it in the past and their tested solution is in a repository.

Most current interfaces to these repositories are at best primitive and at worst unusable. These interfaces provide the set of files that "match" a given set of keywords. While this type of interface works well for web-based information, it does not work as well for code since keywords are not the best search interface for code. The immediate search results are typically shown initially as code snippets containing the keywords along with links that provide one click access to the file, to a list of files in the package, or to the project root. The code snippets are rarely enough to tell whether the returned code is appropriate or not, and the files, presented as a whole and without supporting files or other information, are difficult to navigate and investigate. Using code from the repository is also problematic. The programmer generally will want to extract a minimal amount of code that can be used standalone.

The goal of our work is to develop a better user interface for exploring large software repositories with the goal of finding and extracting code relevant to a particular task. In doing so, we wanted to combine the results from multiple search requests; let searches be easily extended to include other relevant files in a project; provide full navigation capabilities on the returned code; let the returned code be compiled and edited; and export the resultant code. We achieve these aims by combining an integrated development environment front end with a code search back end. Our demonstration will show how the system, REBUS (for Repository Bubbles), works.

This work is supported by the National Science Foundation grant CCF1130822. Additional support has come from Microsoft and Google.

RELATED WORK

The various code repositories, for example GitHub and SourceForge, provide simple mechanisms for browsing. Where you are not certain what you are looking for, they provide a search facility that can be used to find projects or files that contain a given set of keywords. Code search engines such as Ohloh provide a search interface over multiple repositories.

A wide range of code search tool have been developed, mainly based on information retrieval techniques. Examples of recent systems include Sorcerer [2], Codifier [3], Exemplar [6], S^6 [11], and Portfolio [10]. While these efforts concentrate improving the quality of search results, most do not address the issues of displaying the results. There are several efforts that go beyond simple search and file display for browsing repositories. GrepCode provides a more sophisticated interface, letting the user find definitions of a class or method from within the repository by clicking on its name and letting the user compare multiple versions of software [7]. Portfolio [10] displays a graph of the search results showing their relationship and their relevance. Find-Concept [14] presents a ranked list of call graphs from the results. S^6 returns only code that compiles and that passes a set of user-provided test cases [11].

Several of the search engines provide plug-ins to integrate search within a development environment such as Eclipse. Ohloh's Eclipse plug-in, for example, lets the user search for a method given its name and signature. GrepCode's plug-in will search for classes that Java lacks the source for. Other search engines facilitate code search from within the environment. For example [1,8,9,13] let the programmer define a dummy routine and test cases and search for code that will pass the test cases. Microsoft recently released Bing Code Search for C# as a plug-in for Visual Studio that searches code blogs.

Code Bubbles [4,5] redesigns the user interface to programming, making the environment conform to the programmer's working model. Code Bubbles presents software fragments in fully manipulatable interface elements called bubbles in order to provide an intuitive arrangement of working sets. Code Bubbles includes features to simplify and support navigation. It provides a fast search facility that can be used to look at the code hierarchically or to quickly find classes and methods. It supports popping up bubbles for definitions and references by pointing at a name and either hitting a function key. It provides typical environmental facilities to show errors and quickly navigate to them.

Our Solution

Our goal is to provide a better environment for the programmer to explore, identify, and extract relevant code for a particular application from existing repositories. We wanted to make all the navigation features offered by development environments available while browsing repositories. We also wanted to offer an environment that would provide the feedback and editing necessary for the programmer to extract and reuse code from the repository. We achieve these goals by combining the Code Bubbles environment with a back-end that accesses multiple search engines.

Code Bubbles typically runs as a separate tool on top of Eclipse using a message-based plug-in mechanism [12]. It includes a small Eclipse plug-in which connects to a message bus that the main environment talks to. Integration is achieved using command messages from Code Bubbles to Eclipse and informational messages from Eclipse to Code Bubbles. Code Bubbles uses Eclipse to handle file access and editing, searching, compilation, and execution.

Our approach replaces Eclipse in Code Bubbles with a repository browsing back end. This back end handles the same basic set of commands that the Eclipse plug-in does except for those related to debugging.

The back end supports a set of additional search-oriented commands including initiating a code search against a set of repositories given a set of keywords; returning the next set of results from a prior code search; augmenting an existing file returned from code search by including other files from same package or project that might be needed for understanding or use of the code; searching for the definitions of a method or type within a given project; marking a package, file, or section of a file as acceptable; and outputting all acceptable definitions to a given directory.

The back end uses a plug-in technology to support multiple search back ends, with the current set including Ohloh, Github, and GrepCode. The interface takes a set of keywords and the page number (for looking at additional sets of results), and uses the specific plug-in to initiate a search for those keywords and that result page. When the result page comes back, it uses the plug-in to extract the file URLs from the resultant HTML. Then it requests those URLs and uses the plug-in, if necessary, to extract the actual code from the result. Searching for packages and systems is handled by converting the request into an appropriate keyword request. The architecture of the search interface makes it easy to add additional search engine plug-ins. The GrepCode plug-in, for example, was added in under two hours.

To extend search results byeond a single file, the back end does a package search. It first finds the package name from the file in question, and then starts a code search using the search engine that originally returned the file looking for other files in the same package using a search string based on the package name. It goes through the results returned to check if the new files are in the same package, and adds them as active files if so. Multiple result pages are considered to ensure that all files are found. To extend the results beyond the package, the back end does a search for other files in the same project that have related package names (i.e. share a common prefix), and adds those files to the project by scanning the current files from a project to get a list of packages that are either imported or used in qualified names. For each package that is deemed related, all the files from that package are added to the current project. This is repeated until no new packages are added to the set.

The front end of the overall system is the Code Bubbles environment with very minor modifications, most of which are derived from using a separate set of resource files. The modifications include dropping support for debugging, testing, and version management; additional buttons on the tool bar and context menus to support browsing-specific commands; and a new bubble to support code search.

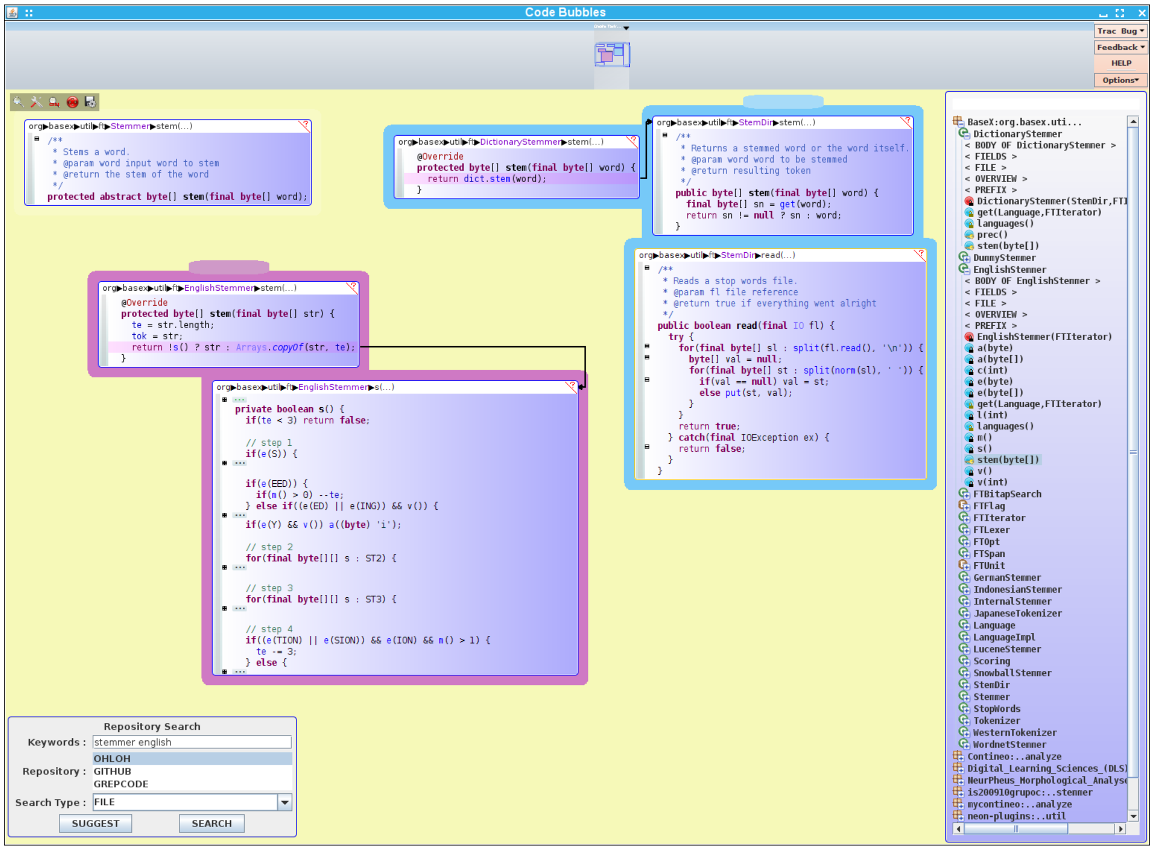
The current implementation is limited to Java files. This is implicit both in the compilation technology incorporated in the back end and in the formatting technology in Code Bubbles. In both cases, the tools are designed so that other languages can be supported and hence the overall browsing environment could be adapted as well.

An Example

Our browsing environment is best illustrated by an example of its potential use shown in Figures 1 and 2. Suppose we were looking for a function that would take a an English word and return the string representing the stem or root of that word.

When we start up our browsing environment, we get a search bubble where we enter the keywords “stemmer english” and select one or more of the available search engines. The search can be done at either the file, package, or system level. File-level searches return a single file; package level searches return, for each file found by the underlying search, all files in its package; system level searches additionally return all files from all related packages.

Once we start the search, the standard Code Bubbles package explorer on the right of the window is populated with the resultant packages. We can use this to look at some of the packages and to select particular methods to view. From the methods, we can used Code Bubbles navigation techniques, for example selecting a call site name and using F3 to bring up the definitions of all methods that could be invoked from the site.



```
org▶basex▶util▶ft▶Stemmer▶stem(...)
/**
 * Stems a word.
 * @param word input word to stem
 * @return the stem of the word
 */
protected abstract byte[] stem(final byte[] word);
```

```
org▶basex▶util▶ft▶DictionaryStemmer▶stem(...)
@Override
protected byte[] stem(final byte[] word) {
  return dict.stem(word);
}
```

```
org▶basex▶util▶ft▶StemDir▶stem(...)
/**
 * Returns a stemmed word or the word itself.
 * @param word word to be stemmed
 * @return resulting token
 */
public byte[] stem(final byte[] word) {
  final byte[] sn = get(word);
  return sn != null ? sn : word;
}
```

```
org▶basex▶util▶ft▶StemDir▶read(...)
/**
 * Reads a stop words file.
 * @param fl file reference
 * @return true if everything went alright
 */
public boolean read(final IO fl) {
  try {
    for(final byte[] sl : split(fl.read(), '\n')) {
      byte[] val = null;
      for(final byte[] st : split(norm(sl), ' ')) {
        if(val == null) val = st;
        else put(st, val);
      }
    }
    return true;
  } catch(final IOException ex) {
    return false;
  }
}
```

```
org▶basex▶util▶ft▶EnglishStemmer▶stem(...)
@Override
protected byte[] stem(final byte[] str) {
  te = str.length;
  tok = str;
  return !s() ? str : Arrays.copyOf(str, te);
}
```

```
org▶basex▶util▶ft▶EnglishStemmer▶s(...)
...
private boolean s() {
  if(te < 3) return false;

  // step 1
  if(e(S)) {
...

  if(e(EED)) {
    if(m() > 0) --te;
  } else if((e(ED) || e(ING)) && v()) {
...
  if(e(Y) && v()) a((byte) 'i');

  // step 2
  for(final byte[][] s : ST2) {
...

  // step 3
  for(final byte[][] s : ST3) {
...

  // step 4
  if((e(TION) || e(SION)) && e(ION) && m() > 1) {
    te -= 3;
  } else {
...
```

Repository Search
Keywords : stemmer english
Repository : OHLOH
GITHUB
GREPCODE
Search Type : FILE
SUGGEST SEARCH

```
BaseX:org.basex.uti...
  DictionaryStemmer
    < BODY OF DictionaryStemmer >
    < FIELDS >
    < FILE >
    < OVERVIEW >
    < PREFIX >
    DictionaryStemmer(StemDir,FTI
    get(Language,FTIterator)
    languages()
    prec()
    stem(byte[])
  DummyStemmer
  EnglishStemmer
    < BODY OF EnglishStemmer >
    < FIELDS >
    < FILE >
    < OVERVIEW >
    < PREFIX >
    EnglishStemmer(FTIterator)
    a(byte)
    a(byte[])
    c(int)
    e(byte)
    e(byte[])
    get(Language,FTIterator)
    l(int)
    languages()
    m()
    s()
    stem(byte[])
    v()
    v(int)
  FTBitapSearch
  FTFlag
  FTIterator
  FTLexer
  FTOpt
  FTSpan
  FTUnit
  GermanStemmer
  IndonesianStemmer
  InternalStemmer
  JapaneseTokenizer
  Language
  LanguageImpl
  LuceneStemmer
  Scoring
  SnowballStemmer
  StemDir
  Stemmer
  StopWords
  Tokenizer
  WesternTokenizer
  WordnetStemmer
Contineo:..analyze
Digital_Learning_Sciences_(DLS)
NeurPheus_Morphological_Analyse
is200910grupoc:..stemmer
mycontineo:..analyze
neon-plugins:..util
```

Figure 1. Result of expanding the first alternative and then investigating two of the potential stemmer classes using Code Bubbles browsing technology. The bubble in the lower left is the search bubble where the user selects the keywords and the search engines to be used. The bubble on the right of the display is the package explorer where the results of the search are displayed. The code bubble in the upper left is abstract method that was chosen first. The three code bubbles to its right show the exploration of the DictionaryStemmer class. The two code bubbles to the left and below show the exploration of the EnglishStemmer class.

Exploring the immediate returned results is generally the first step in the process. The user might quickly determine that some results are not relevant and that other results are essentially duplicates. Our tool lets the user easily eliminate any such results. If more results are desired, the user can easily extend the search to the next set of results, can try the same search against a different set of search engines, or can initiate a new search using a different set of keywords. All these actions will be immediately reflected in the set of results shown in the package explorer.

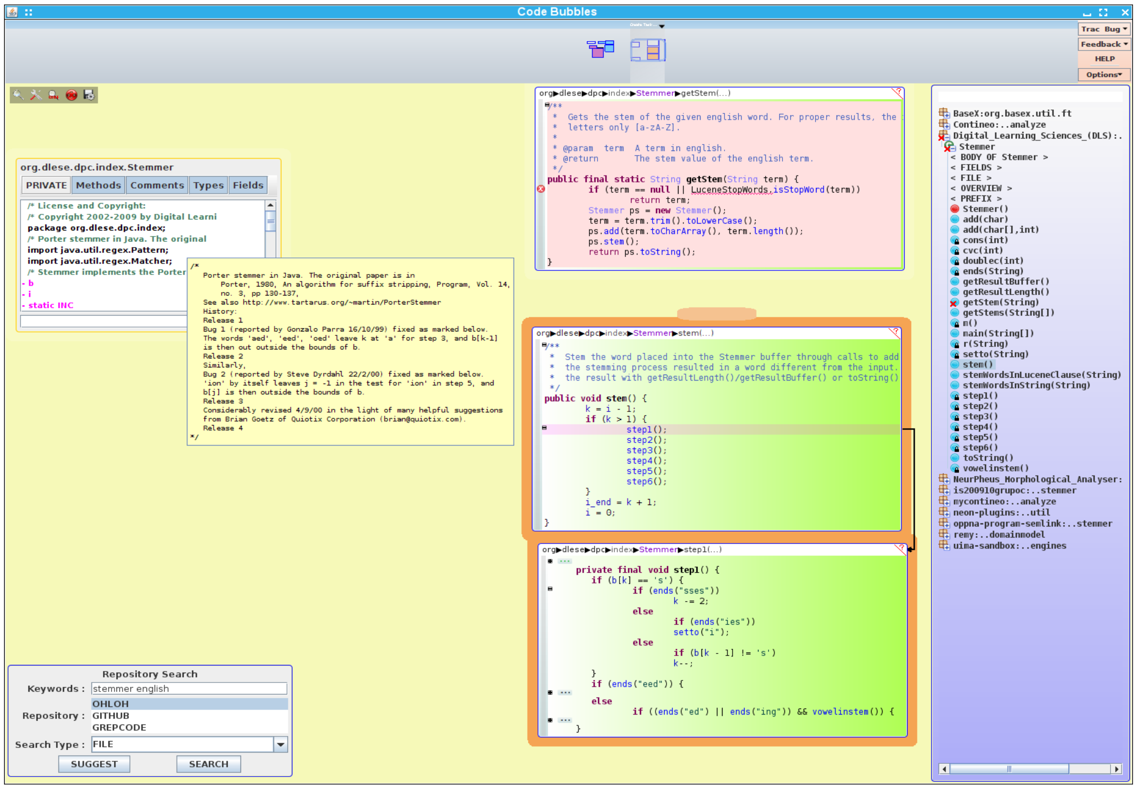
The first result in this case is an abstract class representing a stemmer, which the user notices right away when displaying the *stem* method. In order to understand if the actual implementation of the class might be appropriate, the user requests the system expand the result to include all other classes in the package. If necessary, these results could be further expanded to include all related packages in the project. Doing the package expand in this case shows several potential stemmers, including *DictionaryStemmer* and *EnglishStemmer*. Using the environment to explore these more, shows that the first needs to read a data file which doesn't seem to be available. The second looks hopeful, however. The result so far is illustrated in Figure 1.

Looking at the returned files in the package explorer, shows another result that might be promising in that it seems to be a complete stemmer implementation. The user can bring up an overview window on the class to see the file structure, read the comments, and see the primary methods at a glance. This seems to indicate that the class doesn't need any outside classes and would be more suitable to incorporation into their code. To check this, the user can request compilation. This shows that there is a single error, and the user can quickly find and bring up the method with the error. A quick read of the code shows that the request involves the use of an outside class to check for stop words, but that the check is only for performance, ensuring that these are not stemmed. This is shown in Figure 2.

At this point, the user edits the displayed code to remove the external check, removing the error. Once satisfied, the user marks the file as acceptable by right clicking in the package explorer. This shown by a green mark on the file. Finally, the user can use the tool bar on the upper left to export the edited accepted solution for their use.

CONCLUSION

The tool in its current form serves as a front end for browsing open source repositories where there is an avail-



```
/* License and Copyright:
/* Copyright 2002-2009 by Digital Learni
package org.dlese.dpc.index;
/* Porter stemmer in Java. The original
import java.util.regex.Pattern;
import java.util.regex.Matcher;
/* Stemmer implements the Porter
- b
- i
- static INC
```

```
/*
   Porter stemmer in Java. The original paper is in
       Porter, 1980, An algorithm for suffix stripping, Program, Vol. 14,
       no. 3, pp 130-137,
   See also http://www.tartarus.org/~martin/PorterStemmer
   History:
   Release 1
   Bug 1 (reported by Gonzalo Parra 16/10/99) fixed as marked below.
   The words 'aed', 'eed', 'oed' leave k at 'a' for step 3, and b[k-1]
   is then out outside the bounds of b.
   Release 2
   Similarly,
   Bug 2 (reported by Steve Dyrdahl 22/2/00) fixed as marked below.
   'ion' by itself leaves j = -1 in the test for 'ion' in step 5, and
   b[j] is then outside the bounds of b.
   Release 3
   Considerably revised 4/9/00 in the light of many helpful suggestions
   from Brian Goetz of Quiotix Corporation (brian@quiotix.com).
   Release 4
*/
```

```
org▶dlese▶dpc▶index▶Stemmer▶getStem(...)
/**
 *  Gets the stem of the given english word. For proper results, the
 *  letters only [a-zA-Z].
 *
 * @param  term  A term in english.
 * @return       The stem value of the english term.
 */
public final static String getStem(String term) {
        if (term == null || LuceneStopWords.isStopWord(term))
                return term;
        Stemmer ps = new Stemmer();
        term = term.trim().toLowerCase();
        ps.add(term.toCharArray(), term.length());
        ps.stem();
        return ps.toString();
}
```

```
org▶dlese▶dpc▶index▶Stemmer▶stem(...)
/**
 *  Stem the word placed into the Stemmer buffer through calls to add
 *  the stemming process resulted in a word different from the input.
 *  the result with getResultLength()/getResultBuffer() or toString()
 */
public void stem() {
        k = i - 1;
        if (k > 1) {
                step1();
                step2();
                step3();
                step4();
                step5();
                step6();
        }
        i_end = k + 1;
        i = 0;
}
```

```
org▶dlese▶dpc▶index▶Stemmer▶step1(...)
 ...
     private final void step1() {
        if (b[k] == 's') {
                if (ends("sses"))
                        k -= 2;
                else
                        if (ends("ies"))
                        setto("i");
                else
                        if (b[k - 1] != 's')
                        k--;
        }
        if (ends("eed")) {
 ...
        else
                if ((ends("ed") || ends("ing")) && vowelinstem()) {
 ...
     }
```

```
BaseX:org.basex.util.ft
Contineo:..analyze
Digital_Learning_Sciences_(DLS):.
 Stemmer
  < BODY OF Stemmer >
  < FIELDS >
  < FILE >
  < OVERVIEW >
  < PREFIX >
  Stemmer()
  add(char)
  add(char[],int)
  cons(int)
  cvc(int)
  doublec(int)
  ends(String)
  getResultBuffer()
  getResultLength()
  getStem(String)
  getStems(String[])
  m()
  main(String[])
  r(String)
  setto(String)
  stem()
  stemWordsInLuceneClause(String)
  stemWordsInString(String)
  step1()
  step2()
  step3()
  step4()
  step5()
  step6()
  toString()
  vowelinstem()
NeurPheus_Morphological_Analyser:
is200910grupoc:..stemmer
mycontineo:..analyze
neon-plugins:..util
oppna-program-semlink:..stemmer
remy:..domainmodel
uima-sandbox:..engines
```

Figure 2. A view while exploring the second alternative. Here we first brought up the overview bubble for the class in the upper left and two code bubbles in the bottom center that show the basic stem method and one of the items it calls. Next we built this project, resulting in the error flag in the package explorer. Clicking on that method brings up the code bubble at the top center showing the error. The user can edit this file to remove the error, accept the resultant class using the package explorer and then use the export button on the tool bar at the upper left to save the edited result for their use. The yellow window in the left center is a tool tip showing the contents of a comment from the overview bubble.

able search engine. This tool is implemented as part of the standard Code Bubbles environment. It is available in both source and binary form from the Code Bubbles web site, http://www.cs.brown.edu/people/spr/codebubbles. The tool is run by starting Code Bubbles with the *-rebus* option. For more information including a video, multiple figures from the above example, installation and running instructions, see **http://www.cs.brown.edu/people/spr/codebubbles/rebus.html**.

REFERENCES

1. Marat Akhin, Nikolai Tillmann, Manual Fahndrich, Jonathan de Halleux, and Michal Moskal, "Search by example in touch develop: code search made easy," *Proceedings SUITE 2013*, pp. 5-8 (June 2012).
2. Sushil Bajracharya, Joel Ossher, and Cristina Lopes, "Sourcerer: an infrastructure for large-scale collection and analysis of open-source code," *Science of Computer Programming* **79** pp. 241-259 (2014).
3. Andrew Begel, "Codifier: a programmer-centric search user interface," *Workshop on Human-Coputer Interaction and Information Retrieval*, (October 2007).
4. Andrew Bragdon, Steven P. Reiss, Robert Zeleznik, Suman Karumuri, William Cheung, Joshua Kaplan, Christopher Coleman, Ferdi Adeputra, and Joseph J. LaViola, Jr., "Code bubbles: rethinking the user interface paradigm of integrated development environments," *ACM/IEEE International Conference on Software Engineering 2010*, pp. 455-464 (2010).
5. Andrew Bragdon, Steven P. Reiss, Robert Zeleznik, Suman Karumuri, William Cheung, Joshua Kaplan, Christopher Coleman, Ferdi Adeputra, and Joseph J. LaViola, Jr., "Code bubbles: a working set-based interface for code understanding and maintenance," *Proceedings SIGCHI Conference on Human Factors in Computing Systems*, pp. 2503-2512 (2010).
6. Mark Grechanik, Chen Fu, Qing Xie, Collin McMillan, Denys Poshyvanyk, and Chad Cumby, "A search engine for finding highly relevant applications," *Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering*, (May 2010).
7. GrepCode, "GrepCode Home Page," *http://grepcode.com*, (2013).
8. Werner Janjic, Dietmar Stoll, Philipp Bostan, and Colin Atkinson, "Lowering the barrier to reuse through test- driven search," *SUITE,09*, pp. 21-24 (May 2009).
9. Otavio Lemos, Sushil Bajracharya, Joel Ossher, Paulo Masiero, and Cristina Lopes, "Applying test-driven code search to the reuse of auxiliary functionality," *Proceedings ACM Symposium on Applied Computing*, pp. 476-482 (2009).
10. Collin McMillan, Mark Grechanik, Denys Poshyvanyk, Qing Xie, and Chen Fu, "Portfolio: finding relevant functions and their usage," *Proceeding of the 33rd International Conference on Software engineering*, (May 2011).
11. Steven P. Reiss, "Semantics-based code search," *International Conference on Software Engineering 2009*, pp. 243-253 (May 2009).
12. Steven P. Reiss, "Plugging in and into Code Bubbles," *Proceedings Workshop on Developing Tools as Plug-ins 2012*, pp. 55-60 (June 2012).
13. Steven P. Reiss, "Integrating S6 code search and Code Bubbles," *3rd International Workshop on Developing Tools as Plug-ins*, pp. 25-30 (May 2013).
14. David Shepherd, Zachary P. Fry, Emily Hill, Lori Pollock, and K. Vijay-Shanker, "Using natural language program analysis to locate and understand action- oriented concerns," pp. 212-224 in *Proceedings of the 6th international conference on Aspect-oriented software development*, (2007).